XML 정규화 알고리즘 구현

박기식* · 조인준**· 정회경**

An Implementation of the Canonical XML Algorithm

Ki-shik Park* · In-June Jo** · Hoe-Kyung Jung**

요 약

현재 XML이 전자 상거래 시장에 널리 수용하여 사용되고 있다. 그러나 XML 문서는 논리적으로 동일하나 물리적으로 여러 다른 형태가 존재할 수 있어, XML 디지털 서명과 같은 물리적 형태로써 유효성을 판단하는 응용프로그램에서는 문제점이 발생할 수 있다. 따라서 이런 단점을 해결하기 위해 W3C에서는 논리적으로 동일한 XML 문서를 물리적으로 동일하게 변환시키도록 XML 정규화(Canonical XML) 알고리즘을 제안하여 사용하도록 권고하고 있다.

본 논문에서는 W3C에서 권고한 XML 정규화 알고리즘을 수행하는 시스템을 설계 및 구현함으로써, 좀 더 정교하고 정규화된 문서로 변형하여 W3C 표준을 따르는 다른 응용 시스템과의 상호 운용이 가능하다. 또한 웹 서비스를 위한 전자서명 시스템에서의 사용이 용이할 뿐만 아니라, 웹 서비스 상호 운용성을 위한 XML 문서 교환 시 물리적 동일성이 요구되는 여러 시스템에서의 사용이 용이할 것으로 사료된다. 뿐만 아니라 국제적 인코딩 스킴과 국내 인코딩 스킴인 EUC-KR과의 변환기능을 추가함으로써 국내 실정에 맞는 XML 정규화 알고리즘이 될 것이며, 이는 국제적 상호 운용성 확보의 기반 기술이 될 것이다.

ABSTRACT

These days, XML is accepted and used to e commerce market broadly. But by reason of XML document can exist same form logically but several other forms physically, several problems can happen in application that judge effectiveness as physical form such as XML digital signature. Therefore, it is recommending to propose and use Canonical XML algorithm to change identical XML document physically equally logically in W3C to solve this problems.

We implemented system that run Canonical XML algorithm that suggested in W3C that can change to more elaborate regular document. Thus, interoperable with other application that takes W3C recommendation Also, as well as use in digital signature system for web service is useful, use in several system that physical identify is required when it exchanges XML document for web service interoperability are considered to be valuable. Moreover, Adding the transformation ability between universal encoding scheme and EUC-KR that is internal encoding scheme should be Canonical XML Algorithm that is suited to internal circumstances, and this should be a foundation technique of international interoperability confirmedness.

키워드

XML, XML 디지털 서명, XML 정규화, 웹 서비스 상호 운용성

I. 서 론

최근 인터넷이 활성화되고 사용자가 급속도로 증가하는 추세에 맞추어 기존의 기업들이 현재 갖고 있던 상거래 시장에서 인터넷을 통한 시장 확장

*한국전자통신원구원 **배재대학교 컴퓨터공학과

접수일자 : 2003. 9. 25

을 목표로 전자 상거래에 관심이 모아지고, 각 기업들은 독자적인 전자 상거래 시스템 구축에 많은 투자를 하였다. 이에 발맞추어 차세대 웹 문서인 XML(eXtensible Markup Language)[1]을 전자 상거래 시장에서 수용하게 되었으나 전자 상거래의 특성상 문서의 신뢰와 투명성을 요구하게 되었고, 요구사항을 해결하기 위하여 여러 업체들은 각각 자신들의 독자적인 디지털 서명을 사용하게 되었으나, 다른 시스템들과 상호 운용에 있어서 문제점이 발생하게 되었다. 이런 이유로 XML의 장점을 충분히 활용하지 못하고 있는 것을 느낀 XML 사용자들은 공개적으로 사용할 수 있는 XML 디지털 서명을 요구하게 되었고, 이에 맞추어 W3C(World Wide Web Consortium)에서 XML Signature 표준 명세[2]를 제안하고 사용하도록 권고하고 있다. 이 때 서명할 문서를 표현하기 위한 4가지 변환 알고리즘과 송수신되는 문서의 무결성을 보장하기 위하여 따로 XML 정규화 명세[3]를 제정하여 위의 네 가지 변환 알고리즘과 더불어 사용하도록 권고하고 있다. 더욱이, XML 정규화 알고리즘은 국제적으로 여러 업체(IBM의 XML Security Suite for JAVA (XSS4J)[4], Baltimore사의 X/Secure[5], IAIK의 XML Signature Library (IXSIL)[6])에서 XML 전자 서명 시스템과 함께 개발 중에 있지만, 국내 실정에 맞는 EUC-KR 인코딩 스킴을 지원하는 정규화 시스템이 필요하다.

이에 본 논문에서는 일반 XML 문서를 받아들여 XML 정규화 명세에 따라 정형화된 XML 문서로 변환시키는 XML 정규화 모듈을 설계 및 구현하였다. 이는 XML 디지털 서명 뿐만 아니라 XML을 이용한 메시지 교환을 위한 응용분야에서 XML 문서의 물리적 동일성을 보장하는데 사용될 수 있다. 또한 UTF-8, UTF-16과 EUC-KR의 상호 변환 기능을 추가함으로써 국제적 상호 운용성을 보장할 수 있다.

II. 디지털 서명과 XML 정규화

본 장에서는 논리적 동일성과 물리적 동일성에 대한 설명을 하고 서로 다른 플랫폼이나 프로그래밍 언어를 사용하는 시스템에서 물리적으로 변형될 수 있는 XML 문서를 정형화 시키는 W3C의 Canonical XML에 대하여 설명한다.

2.1 논리적 동일성과 물리적 동일성

논리적 동일성이란 XML문서의 의미가 동일함을 뜻하고 물리적 동일성이란 공백문자를 포함한 모든 문자열이 같은 문서를 말한다. 즉, 논리적으로는 동일하지만 물리적으로 상이하다는 것은 XML 문서 표현의 자율성에 의하여 의미는 같지만 문자열이 다른 문서를 뜻한다. 다음 두 예는 논리적으로는 동일하지만 물리적으로 다른 구조를 가진다.

```
<name a ="1" b="2" c="3"></name>
```

```
<name c='3' b='2' a='1'/>
```

먼저, 자식노드를 가지고 있지 않은 엘리먼트는 시작 태그와 끝 태그로 작성할 수 있지만 두 번째 예처럼 빈 엘리먼트로 작성할 수도 있다. 다음으로는 속성 값의 표현에서 이중 따옴표를 사용한 것을 단일 따옴표로 사용할 수 있고 속성들의 순서를 바꾸어도 XML 문서의 의미전달에 있어서는 아무런 지장이 없다. 이 몇 가지 예에서 본 바와 같이 XML 문서 표현의 자율성이 존재한다. 그러나 이러한 표현의 자율성은 XML 전자서명과 같은 문서의 물리적 구조를 중요시 하는 응용프로그램에서는 치명적 문제를 발생시킨다. 다음의 예는 위의 두 문서를 SHA-1 해쉬 알고리즘을 통하여 생성된 digest이다.

```
zllvxonF84RYwgaWCt7hMOL3STo=
```

```
FqPFgYm5HkfoDEJnxJ6Lt/oqfZU=
```

두 digest값을 비교하여 그 값이 일치하여야 전자서명이 확인될 수 있기 때문에 위의 두 예는 전

자서명이 깨지게 된다. 따라서 여러 가지 표현이 존재할 수 있는 XML 문서를 일관된 형식으로 변환시켜 물리적 동일함을 유지시키는 정규화 알고리즘이 필요하게 된 것이다. 정규화 알고리즘은 W3C Canonical XML 명세를 따른다. 이 명세에 따른 위 두 문서는 다음과 같이 변환된다.

<name a="1" b="2" c="3"></name>
<name a="1" b="2" c="3"></name>

즉, 두 문서가 물리적으로 동일한 형태로 변환되는 것이다. 이에 따른 digest 값은 각각 다음과 같다.

zIlvxonF84RYwgaWCt7hMOL3STo=
zIlvxonF84RYwgaWCt7hMOL3STo=

따라서 생성된 digest값을 비교하는 전자서명에서는 문제점이 발생하지 않는다.

2.2 XML 정규화

웹 서비스 클라이언트와 서버가 서로 연결하고자 하는 시스템에 대하여 상호간에 정보를 주고받는데 지장이 없어야 한다. 특히, 서로 다른 플랫폼이나 프로그래밍 언어를 사용하는 환경의 시스템에서도 그러하다. 가능한 변환들은 XML 서명 권고안에서 정의한 네 개의 타입과 동일하다. 이 네 개의 타입은 XML 프로세스의 변환, XML 파서의 변환, UTF 변환, 네임스페이스 변환[7][8]이다.

표현의 유연성을 가진 XML 문서를 물리적으로 동일한 문서로 만들기 위해 W3C에서는 XML 정규화 알고리즘을 권고하였다. XML 문서를 정규화 함으로써 현재의 웹 서비스 환경에서 비즈니스 파트너와의 관계를 유지하면서 서로 다른 구조로 작성된 XML 문서에 대해 상호 운용성[9]을 보장 할 수 있다.

현재 디지털 전자 서명에서는 특정 키와 알고리즘을 이용하여 전자 서명을 계산한다. 다음으로 데이터와 전자서명을 저장하거나 다른 장소로 전송한다. 그 뒤, 데이터를 불러오거나 전송 받은 다음, 데이터가 전송할 당시의 원본과 같은 것인지를 검사하기 위해 전자 서명을 확인한다. 전자 서명을 계산한다는 것은 2진수 8비트로 이루어진 바이트들로 서명한다는 말이다. 만일 서명 될 데이터가 이미 고정된 바이트로 이루어져 있다면 큰 문제가 되지 않을 것이나, 실제로 데이터가 XML과 같은 평문(plan text)으로 되어있다면, 식별자나 서명자가 이용했던 것과 같은 옥텟(octet)열을 이용하여 서명을 확인할 수 있다는 보장을 하기가 힘들다. 예를 들면, 텍스트가 하나의 응용프로그램에서 다른 응용프로그램으로 전송될 때 물리적 변화(행 종료문자, 공백처리, 인코딩 등의 변화)가 일어날 수 있다. 이런 변화가 생긴 텍스트들은 원래의 텍스트와 다른 바이트열로 표현되기 때문에, 그 텍스트의 서명이 확인되지 못한다. 따라서, W3C에서 XML 정규화 명세를 통하여 다음과 같은 규칙을 권고하고 있다.

- 속성값의 표준화
- 문자와 파싱된 엔터티 참조는 대체
- XML 선언과 DTD 제거
- 공백 엘리먼트는 시작 태그와 종료태그의 쌍으로 대체
- 문서 엘리먼트의 외부 공백과 시작 태그와 종료 태그에 있는 공백을 표준화
- 문자 내용의 모든 공백은 유지
- 속성값 구분자는 이중 따옴표로 대체
- 속성값과 문자 내용의 특수 문자들은 문자 참조로 대체
- 엘리먼트에서 불필요한 네임스페이스제거
- 각 엘리먼트에 디폴트 속성 추가
- 각 엘리먼트와 네임스페이스는 사전적 순서에 의해 정렬
- 문서는 UTF-8로 인코딩한다.
- 행 종료문자는 #xA로 대체한다.

위의 규정들을 통해 응용 프로그램들 뿐만 아니라 사용자에 의해 변형 될 수 있는 의미적으로는 동일하지만 물리적으로 상이할 수 있는 문서를 정규화 할 수 있다.

III. XML 정규화 알고리즘 설계

3.1 전체 시스템 구성도

W3C에서 정의한 XML 정규화 명세에 따르면 입력으로 두 가지 파라미터를 받아들이도록 되어 있다. 첫번째 파라미터로는 주석을 넣을 것인지 결정하는 주석처리 불(Boolean) 값이고 두 번째 파라미터로는 XML 문서를 옥텟 스트림(Octet Stream)으로 받아들일 것인가, 아니면 노드 셋(Node Set)으로 받아들일 것인가를 결정하는 것이다. 따라서 본 모듈의 인스턴스(instance)를 생성할 때 주석처리 불값을 파라미터로 받아들이고 생성한 인스턴스의 직렬화 모듈을 호출할 때, 그 파라미터로 파일 또는 노드셋을 받아들이도록 설계하였다. 그림 1은 전체 처리 흐름도를 UML의 활동 다이어그램으로 표시한다.

그림 1과 같이 인스턴스를 생성할 때 첫 번째 파라미터로 받아들이는 값은 주석 처리 불이다. 이 값에 따라 결과 문서에 주석을 추가할 것인지 아닌지를 결정한다. 그 뒤 Write 클래스를 생성하고 그 파라미터로 변환될 문서의 인코딩 방식을 지정하여 다양한 인코딩 스킴(scheme)에 따라 문서를 변환시킬 수 있다. 기본 값으로 W3C에서 권고한 UTF-8이 지정되어있지만 EUC-KR을 지정하여 국내 실정에 맞는 XML문서로 변환 시킬 수 있다. 두 번째 파라미터로 받아들이는 값은 문서를 받아들이는 형식인데, 그 값이 문서 형식이면 바로 SerializeNode 모듈로 보내지고, 파일 형식으로 받아들였다면 문서 형식으로 변환한 뒤에 보내진다. 이 SerializeNode 모듈은 받아들인 노드의 형식과 7가지 노드의 형식(Document, Element, Namespace, Attribute, Text, Processing Instruction, Comment)을 비교하여 그 중 일치하는 노드에 따라 분기 하는 역할을 수행한다. 분기된 노드는 그에 대응하는 처리 모듈로 보내지게 되고,

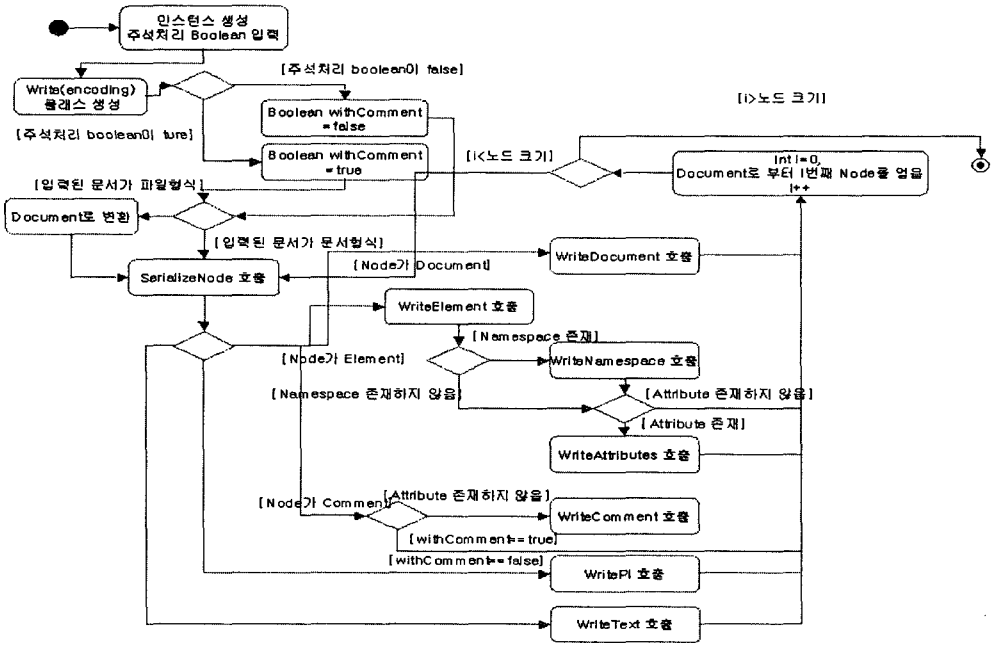

그림 1. 전체 시스템 흐름도
Fig. 1 System flow chart

각 처리 모듈은 DOM을 이용하여 원본 XML 문서로부터 가져온 데이터와 해당 노드에 적절한 텍스트를 혼합하여 임의의 텍스트에 내려 쓰는 형식이다. 각 노드에 대한 처리 모듈은 다음 절부터 상세히 설명한다.

3.2 문서 노드 처리

문서 노드 처리에서는 문서의 가장 상위 노드인 문서 노드를 받아들여 자신이 가진 자식 노드의 수만큼 반복하여 처리한다. 그림 2는 문서 노드 처리를 활동 다이어그램으로 나타낸 것이다. 받아들인 문서의 자식 노드를 순차적으로 직렬화 모듈의 파라미터로 전달한다. SerializeNode 모듈에서는 받아들인 노드의 종류에 따라 그에 적절한 노드 처리로 보낸다.

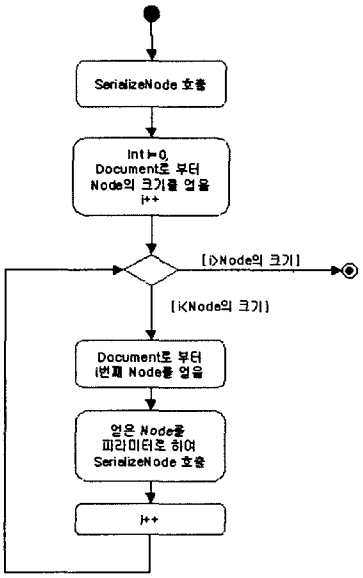

그림 2. 문서 노드 처리
Fig. 2 Processing Document Node

한다. 그림 3은 받아들인 네임스페이스 처리과정을 보인다.

자신의 부모 노드와 같은 네임스페이스를 가진다면 처리하지 않도록 하여 불필요한 네임스페이스를 출력하지 않도록 한다. 이 과정이 없다면 동일한 네임스페이스를 가진 모든 자식 엘리먼트는 동일 네임스페이스 출력결과를 가져온다. 따라서 현재 노드와 부모노드를 비교하여 같은 네임스페이스를 가지고 있다면, 네임스페이스 작성과정을 생략하게 함으로써 이러한 문제를 해결한다.

부모의 네임스페이스와 동일하지 않다는 가정하에 처리 순서를 살펴보면, 네임스페이스를 선언하기 위한 "xmlns"를 작성하고 접두사가 존재한다면 ':'과 접두사를 작성하고 난 뒤 DOM을 사용하여 가져온 네임스페이스 URI(Uniform Resource Identifier)를 작성한다.

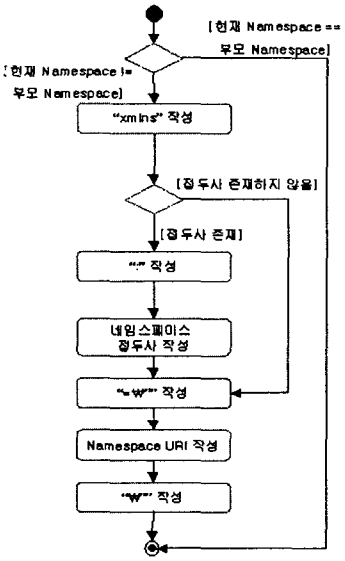

그림 3. 네임스페이스 노드 처리
Fig. 3 Processing Namespace Node

3.3 네임스페이스 노드 처리

네임스페이스 노드 처리는 엘리먼트 노드 처리에서 호출된다. 이 처리 모듈은 정렬된 네임스페이스 노드를 받아들여 순서대로 순차적으로 처리

3.4 엘리먼트 노드 처리

엘리먼트 노드 처리에서는 그 엘리먼트에 속한 네임스페이스 노드와 속성노드를 같이 처리한다.

먼저, MDO(Markup Declaration Open)을 작성한 뒤, DOM으로부터 해당 엘리먼트의 이름을 작

성한다. 다음으로 존재하는 네임스페이스나 속성을 각각 해당하는 스택에 넣고 정렬한 뒤에 꺼내는 순서대로 네임스페이스 노드 처리, 속성 노드 처리로 보내어 처리한다. 네임스페이스나 속성이 존재하지 않는다면 이러한 처리과정을 거치지 않는다. 다음으로 MDC(Markup Declaration Close)를 작성함으로써 시작 태그(Start Tag)에 대한 처리를 마친다. 다음으로 자식 노드가 존재한다면, 자식의 수만큼 반복하여 그 자식 노드를 순차적으로 직렬화 모듈의 파라미터로 보내어 처리하게 된다. 자식 노드가 엘리먼트 노드라면 다시 이 처리과정을 반복할 것이고, 모든 자식 노드들에 대한 처리가 끝났다면 다시 MOD와 문자 '/' 그리고 DOM을 사용하여 가져온 해당 엘리먼트의 이름과 MDC를 작성함으로써 종료 태그(End Tag)에 대한 처리를 마친다. 물론 자식 노드가 존재하지 않는다면 시작 태그와 종료 태그의 쌍만 존재하게 된다. 이로써 공백 태그(Empty Tag)로써 작성될 수 있는 엘리먼트를 시작 태그와 끝 태그의 쌍으로 작성함으로써 물리적 동일함을 유지할 수 있다.

3.5 속성 노드 처리

속성 노드 처리 또한 엘리먼트 노드 처리에서 호출되는 모듈로써 정렬된 속성들을 순차적으로 받아들여 처리한다.

먼저 엘리먼트의 이름이나 네임스페이스 또는 다른 속성과 구분하기 위한 공백을 작성한 뒤 DOM을 이용하여 속성의 이름을 가져와 작성한다. 그 뒤로 속성 값을 처리하는 부분은 명세에서 지정한 &, <, ", #x9, #xA, #xD 문자들을 문자들의 참조로 대치하도록 속성 값을 문자단위로 나누어 비교, 대체한다.

3.6 텍스트 노드 처리

텍스트 노드 처리에서는 명세에서 지정한 <, >, &, &#xD 네 문자를 각 문자의 참조로 대치하도록 DOM을 이용하여 텍스트를 가져와 그 텍스트를 문자 단위로 나누어 비교, 대체한다.

3.7 처리 지시자 노드 처리 및 주석 노드 처리

처리 지시자 노드의 경우 MDO와 '?'를 작성하고 DOM을 이용하여 처리 지시자의 이름과 공백을 작성한 뒤 처리 지시자의 텍스트를 작성하고 '?'과 MDC를 작성한다.

주석 노드의 경우 MDO와 "!--"를 작성하고 역시 DOM을 이용하여 주석 노드로부터 텍스트를 가져와 작성한 뒤 "--"과 MDC를 작성한다.

IV. 구현

4.1 구현 환경

본 시스템은 IBM PC 호환 컴퓨터(Pentium IV 2.4G)에서 Windows 2000 운영체제 하에서 개발 도구로 JBuilder 8.0을 사용하였으며, 개발 언어로는 Java JDK(Java Development Kit) 1.4를 사용하여 앞 장의 설계에 따라 구현하였다. 문서를 생성하고 DOM 객체를 생성하기 위한 XML 파서(Parser)로는 MetaStuff의 DOM4J[10]를 사용하였다.

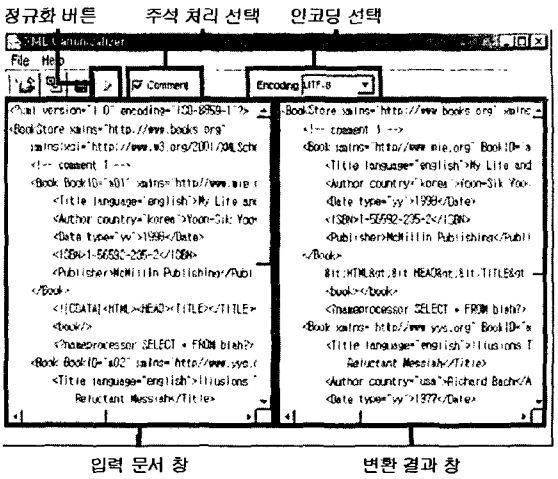

그림 4. 사용자 인터페이스
Fig. 4 User Interface

그림 4는 시험을 위한 사용자 인터페이스 화면이다. 인터페이스의 좌측 창에는 정규화 시킬 원

본 XML 문서를 보여주고 그 상단의 도구막대의 정규화 버튼을 클릭하여 발생하는 이벤트로 정규화 모듈을 호출하여 문서를 정규화 한다. 그 정규화된 결과 화면을 우측 창에 보여줌으로써 원본 문서와 결과 문서를 비교하여 볼 수 있다. 또한 주석을 처리할 것인가에 대한 선택사항을 체크박스 형식으로 선택할 수 있도록 하였다.

4.2 시스템 구현

본 시스템은 XML 문서를 파싱하여 DOM 형태의 결과로 변환해 주기 위한 기존의 XML 파서 부분을 제외하고는 속도 향상 및 외부 모듈로부터의 의존도를 최소화 하여 본 시스템에 해당하는 사용자 인터페이스에서 뿐만 아니라 다른 여러 응용 프로그램에서도 사용이 가능하도록 하였다.

본 시스템은 받아들인 XML 문서를 노드에 따라 분기하기 위한 Serializer.java와 분기된 각각의 노드에 따른 처리를 하기 위한 WriteNode.java, 그리고 시험을 위한 사용자 인터페이스인 Canonicalizer.java 클래스를 구현하였으며, 이에 대한 설명을 표 1에 보인다.

표 1. 구현된 클래스
Table. 1 Implemented Class

클래스명	설명
Serializer.java	구동시키기 위한 클래스로서 XML 문서를 받아들여 각 노드별로 분기하여 WriteNode.java 내의 각 노드에 해당하는 메소드를 호출
WriteNode.java	각 노드에 대한 실제 처리를 하는 클래스로서 실제 문서에 필요한 문자와 텍스트를 출력
Canonicalizer.java	모듈을 시험하기 위한 사용자 인터페이스를 구성하는 클래스로서 Serialize.java의 인스턴스를 생성하여 사용

4.3 구현 결과

입력 문서

```
<?xml version="1.0" encoding="UTF-8"?>
<BookStore xmlns="http://www.books.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema instance" xs
i:schemaLocation="http://www.books.org BookStore1.xsd">
    <!—This is Comment-->
    <Book BookID="a01" xmlns="http//www.mie.org">
      <Title language="english">My Life and Times</Title>
      <Author country="korea">Gil-Dong, Hong</Author>
    </Book>
<![CDATA[<HTML><HEAD><TITLE></TITLE></HEAD>
<BODY><P></P></BODY></HTML>]]>
<Book/>
<?nameprocessor SELECT * FROM blah?>
    <Book BookID="a03" Ccc="cc" Aaa="bb" xmlns="http//w
ww.etri.org">
      <Title language="english">The First and Last Freedo
m</Title>
      <Author country="russia">J. Krishnamurti</Author>
    </Book> </BookStore>
```

변환된 문서

```
<BookStore xmlns="http://www.books.org" xmlns:xsi="htt
p://www.w3.org/2001/XMLS chema instance"xsi: schem
aLocation="http://www.books.org BookStore1.xsd">
    <!--This is Comment-->
    <Book xmlns="http//www.mie.org" BookID="a01">
      <Title language="english">My Life and Times</Titl
e>
      <Author country="korea">Gil Dong, Hong</Author>
    </Book>
&lt;HTML&gt;&lt;HEAD&gt;&lt;TITLE&gt;&lt;/TITLE&gt;&lt;
/HEAD&gt;&lt;BODY&gt;&lt;P&gt;
&lt;/P&gt;&lt;/BODY&gt;&lt;/HTML&gt;
<Book></Book>
<?nameprocessor SELECT * FROM blah?>
    <Book xmlns="http//www.etri.org" Aaa="bb" BookID="a03"
Ccc="cc">
<Title language="english">The First and Last Freedom</Titl
e>
      <Author country="russia">J. Krishnamurti</Author>
    </Book> </BookStore>
```

상기 문서는 본 알고리즘 구현을 이용하여 XML 문서를 정규화 시켜 본 결과이다.

위의 두 문서에서 기본 네임스페이스 선언 부분과 CDATA섹션 부분에서는 지면 관계상 개행이 된 부분이 있다. 변환 결과를 살펴보면, XML 선언부가 제거되었음을 볼 수 있고, 시작 태그 내의 네임스페이스 사이에 존재하는 불필요한 공백문자와 개행 문자가 제거되었음을 볼 수 있다. 주석 노드의 경우에는 인터페이스 상단의 With Comment 체크박스에서 주석을 추가하도록 선택하여 결과 문서에는 주석이 포함되었음을 볼 수 있다. 그러나 이 체크박스의 선택을 해제하고 정규화 시킨다면 주석이 포함되지 않는다. 그 다음으로 볼 수 있는 것은 CDATA 섹션 처리의 부분

으로 그 내부 값은 참조되는 엔터티 값으로 대체시켜야 한다는 명세에 따라 이 노드를 일반 텍스트와 같이 취급하여 텍스트 노드 처리에서 처리하여 해당하는 문자들이 대체되었음을 확인할 수 있다. 그 뒤로는 엘리먼트에 존재하는 속성들과

네임스페이스의 순서가 네임스페이스, 속성 순으로 정렬되고 여러 속성들은 알파벳 순으로 정렬되었음을 볼 수 있다.

표 2. 물리적으로 상이한 두 문서의 변환 결과
Table. 2 The conversion result of two documents that have different physical structure

	입력문서	변환된문서
A	<?xml version="1.0"?> <BookStore> <Book BookID="a03" Ccc="cc" Aaa="bb"> <Title language='english'> The First and Last Freedom</Title> <Author country="russia"> J. Krishnamurti </Author> </Book> <Book/> </BookStore>	<BookStore> <Book Aaa="bb" BookID="a03" Ccc="cc"> <Title language="english"> The First and Last Freedom </Title> <Author country="russia"> J. Krishnamurti </Author> </Book> <Book></Book> </BookStore>
B	<?xml version="1.0"?> <BookStore> <Book BookID="a03" Aaa="bb" Ccc="cc"> <Title language="english"> The First and Last Freedom </Title> <Author country="russia"> J. Krishnamurti </Author> </Book> <Book></Book> </BookStore>	<BookStore> <Book Aaa="bb" BookID="a03" Ccc="cc"> <Title language="english"> The First and Last Freedom </Title> <Author country="russia"> J. Krishnamurti </Author> </Book> <Book></Book> </BookStore>

표 2는 논리적으로는 동일하지만 물리적으로 상이한 두 XML문서를 물리적으로 동일한 문서로 변환되었는지를 보여준다. 입력된 두 문서 A, B는 논리적으로는 동일하지만 물리적으로는 상이하다. 그러나 정규화 모듈을 통하여 변환된 두 문서는 물리적으로 동일한 구조를 가진다. 다음 표 3은 변환된 두 문서를 입력으로 하여 SHA-1 해쉬 알고리즘을 통하여 생성된 다이제스트 값이다.

동일한 물리적 구조를 가지면 그 다이제스트 값 또한 동일하기 때문에 위의 두 변환된 문서는 물리적으로 동일하다고 판단할 수 있다. 이렇게 다이제스트 값을 비교하여 봄으로써 물리적 동일성을 판단할 수 있다.

표 3. 물리적으로 동일한 두 문서의 다이제스트 값
Table. 3 The digest value of two documents that have same physical structure

문서	다이제스트 값
A	DEBihsXYIa2ptiQBtoqHw8LPkuE=
B	DEBihsXYIa2ptiQBtoqHw8LPkuE=

또한 XML 정규화 1.0 명세의 예제 중심으로 하며, 다음 표 4는 명세에서 규정한 13가지 규칙이 모두 구현되었음을 보여준다.

표 4. W3C 정규화 명세 준수
Table. 4 Observance of W3C Recommendation

W3C XML 정규화 명세	비고
속성값의 표준화	만족
문자와 파싱된 엔터티 참조는 대체	만족
XML 선언과 DTD 제거	만족
공백 엘리먼트는 시작 태그와 종료태그의 쌍으로 대체	만족
문서 엘리먼트의 외부 공백과 시작 태그와 종료 태그에 있는 공백을 표준화	만족
문자 내용의 모든 공백은 유지	만족
속성값 구분자는 이중 따옴표로 대체	만족
속성값과 문자 내용의 특수 문자들은 문자 참조로 대체	만족
엘리먼트에서 불필요한 네임스페이스 제거	만족
각 엘리먼트에 디폴트 속성 추가	만족
각 엘리먼트와 네임스페이스는 사전적 순서에 의해 정렬	만족
문서는 UTF-8로 인코딩한다.	만족
행 종료문자는 #xA로 대체한다	만족

여러 XML 문서를 시험하여 본 결과 관련연구 2.2에서 제시한 정규화에 필요한 요구사항에 따른 13가지를 만족하였으며 문서의 인코딩 변경 또한 가능하였다. 인코딩 변경의 확인은 문서로부터 인코딩을 스트링 값으로 얻어 출력하여 봄으로써 확인을 하였다.

이렇듯 W3C에서 권고하고 있는 Canonical XML을 따름으로써 이 표준을 따르는 다른 여러 응용프로그램간의 상호 운용성 문제를 해결할 수 있다.

V. 결론 및 향후 연구 방향

최근 웹 서비스에 대한 관심이 높아짐에 따라 웹 서비스의 장점 중 하나인 상호 운용성을 빼놓을 수 없다. 웹 서비스에서 SOAP 메시지 형태는 XML이며, 현재와 같은 형태의 서비스간 XML 문서 교환에서는 다양한 파서의 이용이 가능하기 때문에 상호 운용성 문제가 발생 할 수 있는 문제점을 안고 있다. XML로 데이터를 교환하는 웹 서비스 환경에서는 이기종간 물리적으로 상이한 XML 문서 구조를 가지고 있다면 XML 전자서명과 같이 물리적 형태로써 유효성 여부를 판단하는 응용프로그램 등에서는 치명적인 문제를 야기시킬 수 있다. 이러한 문제점의 해결을 위한 대안으로 XML 문서의 디지털 서명 보안기능을 제공하는 W3C의 권고안인 XML 정규화 변환 알고리즘이 존재하지만 해당 알고리즘을 곧바로 웹 서비스 이용에 적용하는데 따른 관련 연구가 선행되어야 한다. 따라서 본 논문에서는 논리적으로는 동일하지만 물리적으로 상이할 수 있는 XML 문서를 물리적으로 동일하도록 재 구성하여 이러한 문제를 해결하도록 하는 XML 정규화 시스템을 설계 및 구현하였다.

본 논문에서 제안한 시스템의 특징은 XML 정규화 표준 명세를 적용시킴으로써 다양한 XML 파서로 인한 상호 운용성 문제를 해결하였고 작성자에 따라 논리적으로는 동일하지만 물리적으로 상이할 수 있는 XML 문서를 동일한 물리적 구조를 가지도록 함으로써 좀 더 정교하고 정규화된 문서로 변형할 수 있다. 또한 본 시스템을 모듈화하여 생성함으로써 다른 여러 응용프로그램에서도 이 모듈을 사용하여 정규화 시킬 수 있다. 따라서 XML 문서의 정규화를 통하여 웹 서비스를 위한 전자서명 시스템에서의 사용이 용이할 뿐만 아니라, 웹 서비스 상호 운용성을 위해 XML 문서 교환 시 물리적 동일성이 요구되는 웹 서비스의 많은 응용분야에서의 핵심요소 기술로 사용될 수 있으리라 사료된다.

특히, 현재 사용되는 다양한 인코딩 스킴(scheme)과 EUC-KR 인코딩 스킴과의 상호 변환기능을 제공함으로써 국내 실정에 맞는 기반 기술이 되어 국제적인 상호 운용성의 문제점을 극복할 수 있을 것이다.

향후 연구방향으로는 현재 웹 서비스에서 전송 프로토콜로 사용되는 SOAP(Simple Object Access Protocol)[11]을 정규화 시키기 위한 연구[12]가 진행되어야 하며, 또한 XML 디지털 서명 시스템과 연계하여 통합형 웹 서비스 보안 모델[13]에 관한 연구를 진행할 계획이다.

참고 문헌

[1] Extensible Markup Language 1.0, http://www.w3.org/TR/REC-xml

[2] XML Signature Syntax and Processing, http://www.w3.org/TR/2001/PR xmldsig core 20010820

[3] Canonical XML Version 1.0, http://www.w3.org/TR/xml-c14n

[4] IBM, The XML Security Suite, http://www.trl.ibm.com/projects/xml/xss4j/docs/dsig.html

[5] Baltimore, KeyTools XML Introductions, http://www.baltimore.com/keytools/xml/index.html

[6] IAIK, XML Signature Library (IXSIL), http://jce.iaik.tugraz.at/products/04_ixsil/index.php

[7] Blake Dournaee, XML Security, McGraw-Hill, 2002

[8] Donald E. Eastlake III, Kitty Niles, Secure XML, Pearson Education, 2002

[9] Steve Graham, Simeon Simeonov, Toufic Boubez, Doug Davis, Glen Daniels, Yuichi Nakamura, Ryo Neyama, Building Web Services with Java, SAMS, 2002

[10] DOM4J 1.4 API, http://www.dom4j.org/javadoc

[11] SOAP Version 1.2 Part 1: Messaging Framework, http://www.w3.org/TR/soap12 part1

[12] SOAP Version 1.2 Message Normalization, http://www.w3.org/TR/soap12 n11n

[13] Mark O'Neill, Web Services Security, McGraw-Hill, 2003

저자 소개

박기식(Ki-shik Park)

1982년 : 서울대학교 졸업(학사)
1985년 : 서울대학교 행정대학원 졸업(정책학 석사)
1995년: 충남대학교 대학원 졸업(정책학 박사)
2003년: 배재대학교 대학원 컴퓨터 공학과 박사과정
1996년-현재 : 국제전기통신연합(ITU) 표준화자문위원회 Vice-Chairman, 문서처리분과위원회 Chairman
2001년 4월-현재 : 국가과학기술위원회 국책연구개발사업 평가위원
2002년 4월-현재 : W3C대한민국 사무국 사무국장
1985년 - 현재 : 한국전자통신연구원 표준연구센터장
※ 관심분야 : 정보통신 기술 표준화, 정보통신 QoS, 인터넷 프로토콜, R&D 및 기술 정책

조인준(In-June Jo)

1982년 2월: 전남대학교 계산통계학과 졸업(공학사)
1985년 2월: 전남대학교 대학원 전자계산학과 졸업(공학석사)
1999년 8월: 아주대학교 대학원 컴퓨터공학과 졸업(공학박사)
1991년 12월: 전산조직응용기술사
1983년 9월~1994년 2월: 한국전자통신연구원 선임연구원
1994년~현재: 배재대학교 컴퓨터공학과 부교수
※ 관심분야 : 정보보호, 컴퓨터네트워크, 전산조직응용

정회경(Hoe-Kyung Jung)

1985년 : 광운대학교 컴퓨터공학과 졸업(공학사)
1987년 : 광운대학교 컴퓨터공학과 졸업(공학석사)
1993년 : 광운대학교 컴퓨터공학과 졸업(공학박사)
2001~2003 배재대학교 멀티미디어 지원센터장
1994년~현재 : 배재대학교 정보통신공학부 부교수
※ 관심분야 : 멀티미디어문서정보처리, XML, XSL, DRM ,ebXML, Web Services. MPEG-21, SVG